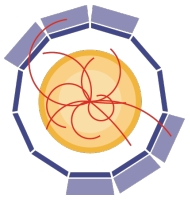

AIDA 2020

Advanced European Infrastructures for Detectors at Accelerators

DDAlign

Alignment Support for the DD4hep Geometry Description Toolkit

Design Document

M. Frank
CERN, 1211 Geneva 23, Switzerland

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

Abstract

Experimental setups in High Energy Physics are highly complex assemblies consisting of various detector devices typically called *subdetectors*. Contrary to the ideal world, where all these components are of perfect shape and at exact positions, existing devices have imperfections both in their shape and their relative and absolute positions. These are described by the alignment parameters.

To still measure the detector response from particle collisions with the highest possible precision, these imperfections are taken into account when converting measured signals to space-points in the measurement devices. This procedure is called *detector alignment*. `DD4hep` does not want to solve the exact problem of the detector alignment itself, but rather support firstly algorithms determining the alignment parameters and secondly support the application which apply the measured alignment parameters and apply them to the ideal geometry for further event data processing.

This document describes how `DD4hep` detector description will accomplish the support for detector alignment data structures, how they will be required by alignment procedures. The design is strongly driven by easy of use; developers of detector descriptions and applications using them should provide minimal information and minimal specific code to achieve the desired result.

Document History		
Document version	**Date**	**Author**
1.0	**01/03/2016**	**Markus Frank CERN/LHCb**
1.1	**30/06/2016**	**Markus Frank CERN/LHCb Revised by workgroup**

Contents

1 Introduction **1**
1.1 Generic Detector Description Model 2
1.2 Detector Element Tree and the Geometry Hierarchy 2
1.3 Alignment Parameters of Detector Components 3
1.4 Iterative Application of Alignments 5
1.5 Procedures to Determine Alignment Parameters 6
1.6 Simulation of Non-Ideal, Real Detector Geometries 6
1.7 Alignment Constants in Multi-Threaded Data Analysis 6

2 The Envisaged DDAlign User Interface **7**
2.1 Access to Alignment Parameters from the Detector Element 7
2.2 Manipulation of Alignment Parameters 9

1 Introduction

This manual should introduce to the `DDAlign` framework. One goal of `DDAlign` is to easily model geometrical imperfections applied to the ideal geometry of detection devices as they are typically used in high energy physics experiments.
To avoid confusion within this document, a few terms need to be defined with respect to detector alignment:

- The *ideal geometry* describes the detector as it was designed. Such a detector is an utopic object, which can never be realized in terms of the placement of the individual components as such.
- The *actual geometry* describes the real detector in the configuration at a given time. This includes all the changes i.e. *deltas* to the *ideal* geometry. These changes are also called the *alignment parameters*. These parameters typically are only valid for a defined time interval.
- *Realignment* defines the procedure to apply a new set of temporary *misalignment parameters* to the ideal geometry. Such a procedure is applied, if a previously applied set of parameters is no longer valid with respect to the event data to be processed. In short *realignment* is necessary if the *actual geometry* of the detector is time dependent.

`DDAlign` formalizes both the access and the application of alignment parameters to the ideal geometry. The possibility to properly describe actual geometries with respect to ideal geometries is essential to understand the detector response to particle collisions and to connect response of geometrical independent areas of the experiment e.g. to one single track.
In this manual we will shortly describe the model used to describe an experiments detector description and then in more detail document the support for alignment with its programming interfaces.

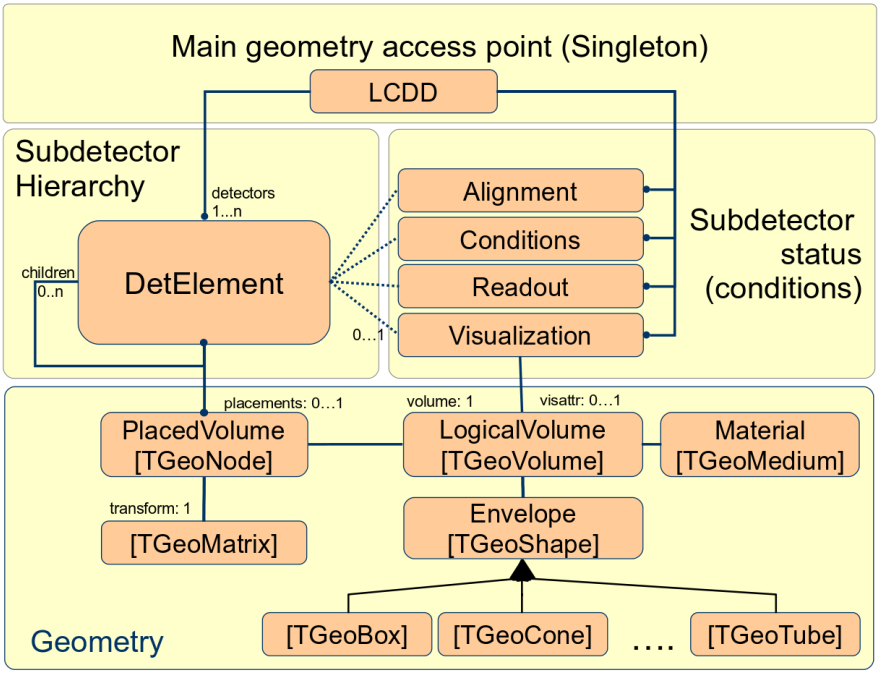

Figure 1: Class diagram with the main classes and their relations for the Generic Detector Description Model. The implementing ROOT classes are shown in brackets.

1.1 Generic Detector Description Model

This is the heart of the DD4hep detector description toolkit. Its purpose is to build in memory a model of the detector including its geometrical aspects as well as structural and functional aspects. The design reuses the elements from the ROOT geometry package and extends them in case required functionality is not available. Figure 1 illustrates the main players and their relationships [1]. Any detector is modeled as a tree of *Detector Elements*, the entity central to this design, which is represented in the implementation by the *DetElement* class [2]. It offers all applications a natural entry point to any detector part of the experiment and represents a complete sub-detector (e.g. TPC), a part of a sub-detector (e.g. TPC-Endcap), a detector module or any other convenient detector device. The main purpose is to give access to the data associated to the detector device. For example, if the user writes some TPC reconstruction code, accessing the TPC detector element from this code will provide access the all TPC geometrical dimensions, the alignment and calibration constants and other slow varying conditions such as the gas pressure, end-plate temperatures etc. The *Detector Element* acts as a data concentrator. Applications may access the full experiment geometry and all connected data through a singleton object called *LCDD*, which provides management, bookkeeping and ownership to the model instances.
The geometry is implemented using the ROOT geometry classes, which are used directly without unnecessary interfaces to isolate the end-user from the actual ROOT based implementation. `DDAlign` allows client to access, manage and apply alignment parameters or smallish changes to the ideal geometry. The mechanism to achieve this is described in the following.

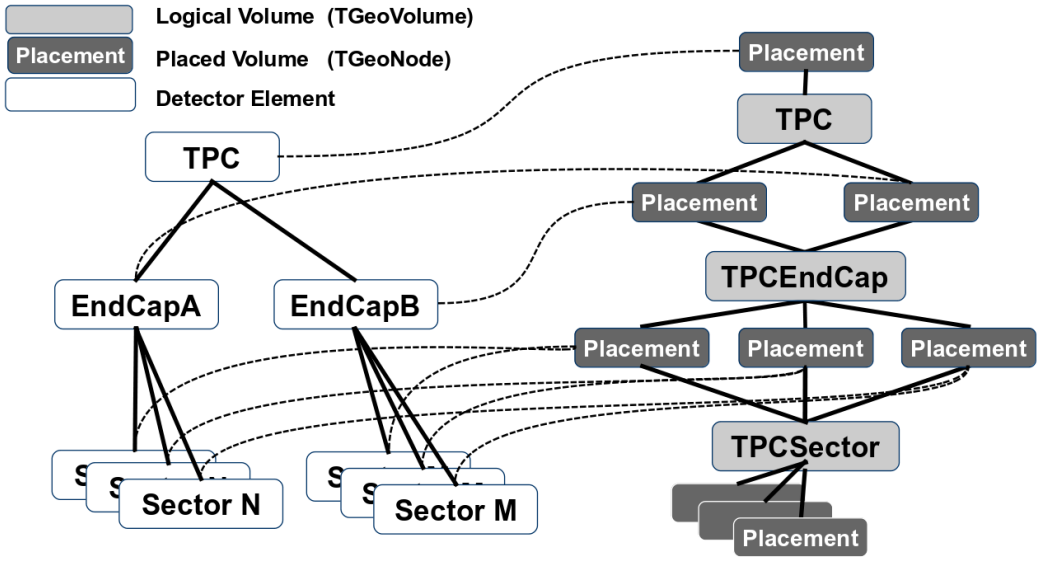

Figure 2: The object diagram of a hypothetical TPC detector showing in parallel the *Detector Element* and the *Geometry* hierarchy and the relationships between the objects.

1.2 Detector Element Tree and the Geometry Hierarchy

The geometry part of the detector description is delegated to the ROOT classes. *Logical Volumes* are the basic objects used in building the geometrical hierarchy. A *Logical Volume* is a shape with its dimensions and consist of a given material. They represent unpositioned objects which store all information about the placement of possibly embedded volumes. The same volume can be replicated several times in the geometry. The *Logical Volume* also represents a system of reference with respect to its containing volumes. The reuse of instances of *Logical Volumes* for different placements optimizes the memory consumption and detailed geometries for complex setups consisting of millions of volumes

may be realized with reasonable amount of memory. The difficulty is to identify a given positioned volume in space and e.g. apply alignment parameters to one of these volumes. The relationship between the Detector Element and the placements is not defined by a single reference to the placement, but the full path from the top of the detector geometry model to resolve existing ambiguities due to the reuse of *Logical Volumes*. Hence, individual volumes must be identified by their full path from mother to daughter starting from the top-level volume.
The tree structure of *Detector Elements* is a parallel structure to the geometrical hierarchy. This structure will probably not be as deep as the geometrical one since there would not need to associate detector information at very fine-grain level - it is unlikely that every little metallic screw needs associated detector information such as alignment, conditions, etc. Though this screw and many other replicas must be described in the geometry description since it may be important e.g. for its material contribution in the simulation application. Thus, the tree of Detector Elements is fully degenerate and each detector element object will be placed only once in the detector element tree as illustrated for a hypothetical Time Projection Chamber (TPC) detector in Figure 2 with an ideal geometry, where no positioning corrections are applied to neither child. It is essential to realize that the geometry tree in an ideal geometry is degenerate contrary to the tree of detector elements.
It should be noted, that alignment parameters may be applied to any volume of the ideal geometry. The alignment only affects the actual position of a volume it is e.g. irrelevant if the volume is sensitive or not.

1.3 Alignment Parameters of Detector Components

Alignment parameters never apply in the same way to *all* placements of the same volume in this hierarchy. Hence, to (re-)align a volume in the hierarchy means to lift a full branch of placements from the top volume down to the element to be (re-)aligned out of this shared hierarchy and apply a correction matrix to the last node. This procedure is illustrated in Figure 3. Re-alignment of volumes may occur at any level. In the above example of a TPC this results in the following effects:

- A realignment of the entire subdetector, i.e. the TPC as a whole, would affect consequently move all contained children with respect to the top level coordinate system. An example is shown in Figure 3 (a). A movement of the subdetector would affect all transformation between local coordinates of any part of the subdetector to the top level coordinate system. Such effects would be visible at all stages of the data processing e.g. when translating signals from particles into global coordinates.
- A realignment of parts of a subdetector affects only the partial subdetector itself and child volumes at lower levels. As in the example, where the entire subdetector is moved, here only the sectors on one side of the TPC would be affected as shown in Figure 3 (b).
- In Figure 3 (c) within one end-cap of the TPC individual sectors may not be positioned at the ideal location (Figure 3 (c) exaggerates: "flying sectors" are a rather rare case in reality). Finally also the sectors itself could be fragmented and be assemblies of other shapes, which are not ideally placed and may need correction.

The origin of the volume misplacements may be many-fold:

- Elements may be weak and assembled parts move due to weak support structures. This is a common problem e.g. for tracking detectors, where heavy and solid structures dramatically influence the measurement result. Misplaced sectors could e.g. be the consequence of a deforming end-cap frame due to the weight of the sectors.
- Environmental conditions such as the temperature may influence the position or the shape of a volume.
- Some of the measurement equipment may be moved from a parking position into a data taking position such as the two halves of the LHCb vertex detector. Whereas the position of the sensors on each half are known to a very high precision, the position of the absolute position of the two halves with respect to the full experiment may change after each movement.

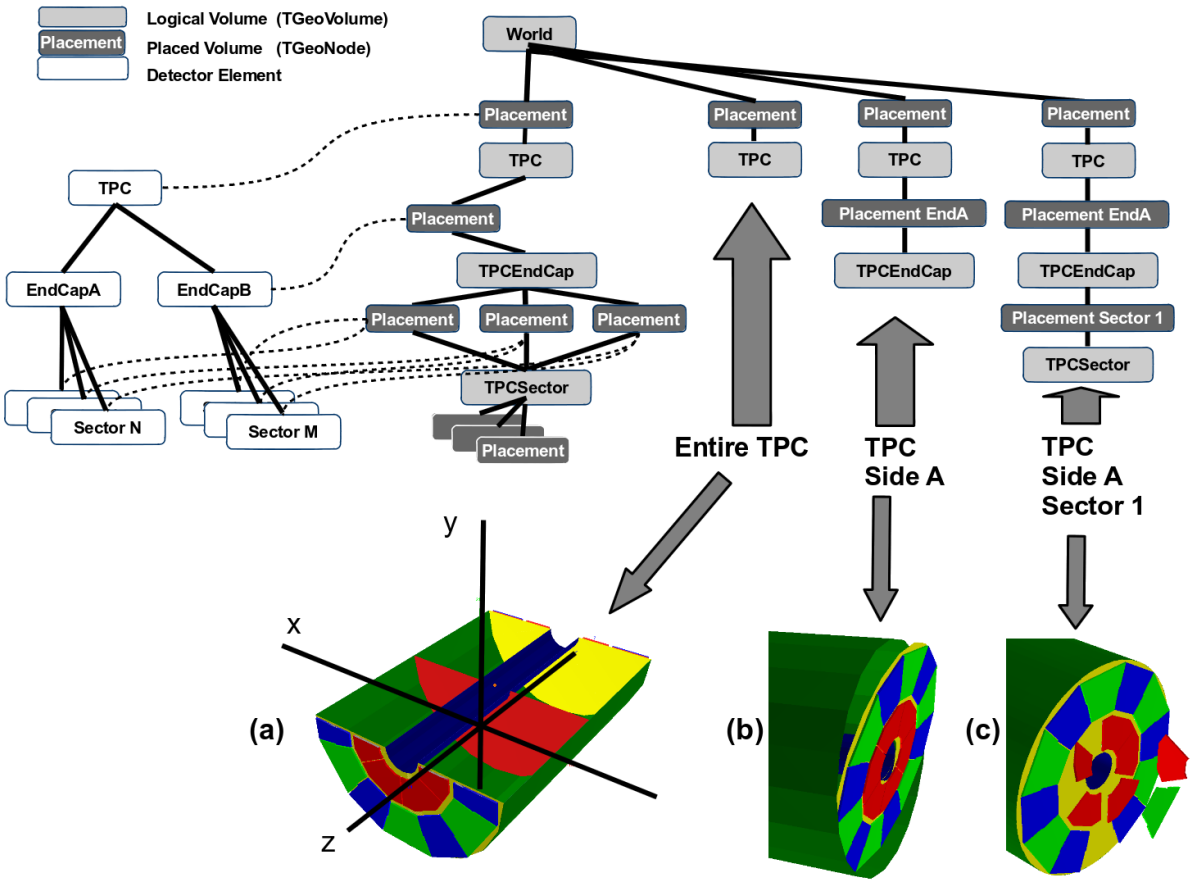

Figure 3: The object diagram of a hypothetical TPC detector showing in parallel the *Detector Element* and the *Geometry* hierarchy and examples of mispositioned detector parts: (a) mispositioned entire subdetector (translation), (b) mispositioned end-cap (tilt) and (c) mispositioned individual sectors within one endcap.

Changes to the volume placement do not only affect sensitive material i.e. detector components with an active readout, but also passive material. The placement of any volume, passive or active, may be corrected using `DDAlign` . The determination of the alignment parameters of passive components however may be more difficult in the absence of located signals resulting e.g. from the traversal of a track.
All effects resulting from such causes obviously need to be corrected in order to fully explore the capabilities of the detection devices and to minimize measurement errors. In general any deviation from the ideal position of a volume can be described by two elementary transformations:

- a translation
- a rotation around a pivot point.

giving a full transformation matrix of the form:

$$T = L * P * R * P^{-1} \tag{1}$$

where

- T is the full transformation in 3D space containing the change to the exiting placement transformation. The existing placement is the placement transformation of the volume with respect to

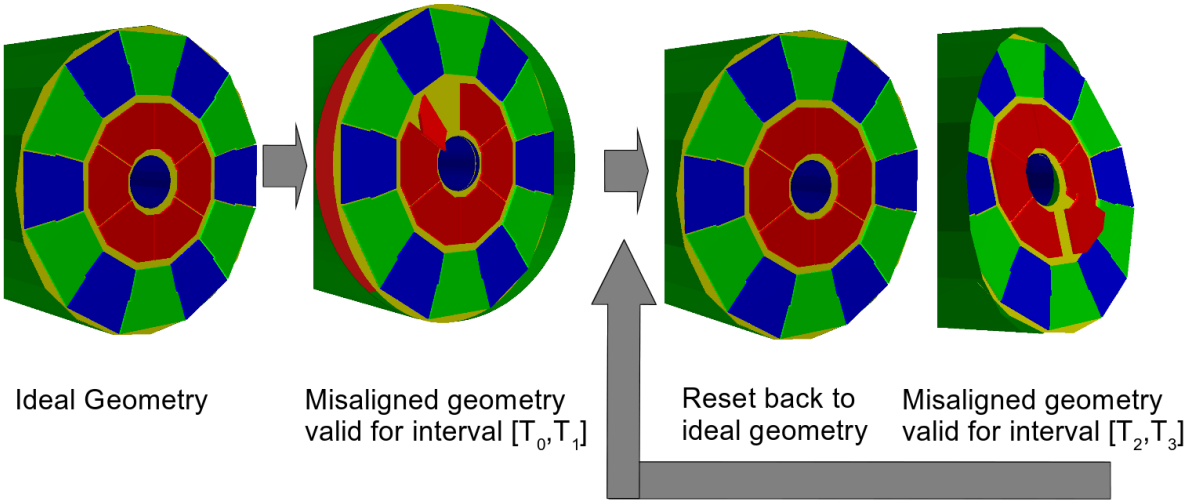

Figure 4: The iterative application of alignment parameters as described in Section 1.4. For each interval of validity ($[T_0, T_1]$, $[T_2, T_3]$, $[T_4, T_5]$, ...) a seperate set of alignment constants is applied to the ideal geometry. The two steps to reset the misaligned geometry back to the ideal geometry and to re-apply a new set of alignment constants may be executed as often as necessary when processing data from particle collisions.

the mother volume.

- L is a translation specifying the position change with respect to the mother volume.
- $P * R * P^{-1}$ describes a rotation around a pivot point specified int he mother volume's coordinate system.
- P is the translation vector from the mother volumes origin to the pivot point. The concept of a pivot point does not introduce a new set of parameters. Pivot points only help to increase the numerical precision.

Most of the changes do not require the full set of parameters. Very often the changes only require the application of only a translation, only a rotation or both with a pivot point in the origin. These simplifications are supported in the user interface described in Section 2.

1.4 Iterative Application of Alignments

In the general case a given set of alignment parameters is not static and may very well change with time. For this reason it is highly important to support not only one single realignment step. Hence, the following scenario is an important use case:

1. Create the ideal detector using an ideal geometry.
2. Apply a set of alignment parameters for a given time interval corresponding to the time a set of particle collisions were collected in the experiment.
3. Process the set of collected particle collisions.
4. Reset the misaligned detector to the ideal.
5. Choose new event data input corresponding to another time interval and restart at item 2.

Graphically this use case is illustrated in Figure 4. In Section 2 the implementation to realize this use case is described.

1.5 Procedures to Determine Alignment Parameters

Typically the determination of alignment parameters requires a starting point which is not necessarily identical to the ideal position of a volume [3]. These volume positions are the result of a survey measurement or the result of internal position measurements of a sub-volume within a sub-detector e.g. on a measurement bench. In the following we call these parameters *survey parameters*. *Survey parameters* default to the ideal volume position if not supplied, alternatively, if set, to the provided position. *Survey parameters* are, like the alignment parameters, provided in terms of *changes* with respect to the ideal position and hence may be treated in a similar way.
The survey parameters are - like alignment parameters - accessible to users through the interface offered by the *DetElement* objects.

1.6 Simulation of Non-Ideal, Real Detector Geometries

It is a standard procedure in high energy physics to at least verify the measured detector response of a given physics process in particle collisions with the expected simulated detector response. For most purposes the simulation of an ideal detector is certainly is sufficient - though not describing the full truth. Sometimes however, the detector geometry must be simulated with a geometry as close to the known geometry as possible.
The simulation of such a geometry with applied alignment parameters can rather easily be realized using using the `DD4hep` , `DDAlign` and the `DDG4` frameworks:

- The ideal geometry is constructed using the standard procedures of `DD4hep` [1].
- Then the alignment parameters are applied.
- Finally the corrected geometry is translated to *Geant*4 [6] using the `DDG4` [4] package. All particle collisions simulated with this translated geometry correspond to the modified geometry including the geometry modifications.

There is a caveat though: The application of alignment parameters can easily create volume overlaps, which are highly disliked by the *Geant*4 runtime. If the above described procedure is applied, it is highly advised to check the resulting geometry for overlaps. Both, *ROOT* [5] and *Geant*4 [6] offer tools to perform such tests.
Simulating displaced geometries was typically not supported by most toolkits in the past. `DDAlign` can offer this feature due to the feature of the *ROOT* geometry toolkit, which actually replaces individual branches of the geometry with truly new placements.

1.7 Alignment Constants in Multi-Threaded Data Analysis

Once alignment constants are applied, the `DD4hep` geometry is read-only. Read-only data structures are by definition thread safe. Multiple threads processing events in parallel will not cause race conditions as long as all events processed in parallel require the same alignment constants. Race conditions may only appear in the event new alignment constants must be applied e.g. when the conditions between subsequent events change.
It is assumed that the hosting framework supports the required actions necessary:

- Drain the event queue
- Apply the new alignment constants
- Re-enable the flow of events to be processed.

Though this is not the only way how to handle multi-threading issues, for data analysis and reconstruction this mechanism is the current baseline unless new requirements emerge from new use-cases of clients.

2 The Envisaged DDAlign User Interface

`DDAlign` implements a machinery to apply and access the alignment parameters describing the difference between an ideal detector given by an ideal geometry and the geometry of the actually built assembly in real life. To ease its usage for the clients and to shield clients from the internals when actually dealing with realigned geometries, a set of helper classes was designed. The access to the alignment parameters in read-only mode was separated from the import or export thereof.
As a basic concept within `DD4hep` any *sizable* detector component can be realigned. *Sizable* as a rule of thumb is anything, which is manufactured as an individual piece and which you may "hold in your hands". Such objects are also described by a *detector element* of type `DetElement`. An example is e.g. a single silicon wafer of a tracking device or the entire tracking detector itself. The access to the alignment parameters is possible from each `DetElement` instance as described in Section 2.1. The interface assumes "planar" alignment parameters i.e. the shape of a given volume does not change [1]. Please be aware that the extensive use of misalignments is highly memory consuming.

2.1 Access to Alignment Parameters from the Detector Element

The *DetElement* class as shown in Figure 1 gives the user access to the alignment structure of type *Alignment* as illustrated in the following example:

```
1    DetElement wafer = ... // Valid handle to a detector element
2    Alignment  wafer_alignment = wafer.alignment();
3    if ( wafer_alignment.isValid() )  {
4        // This wafer's placement differs from the ideal geometry when
5        // alignment parameters are present.
6
7        // Access the misalignment transformation with respect to the parent volume:
8        Transform3D tr = wafer_alignment.toMotherDelta();
9    }
```

The access to details of an invalid alignment object results in a runtime exception. The following calls allow clients to access alignment information from the *DetElement* structure:

```
1      /// Access to the actual alignment information
2      Alignment alignment() const;
3
4      /// Access to the survey alignment information
5      Alignment surveyAlignment() const;
```

The call to *alignment*() return the parameters *applied* to the the existing ideal geometry. The call *surveyAlignment*() returns optional constants used to perform numerical calculations as described in section 1.5.
All functionality of the DetElement, which depends on applied alignment parameters are automatically updated in the event of changes. These are the geometry transformations with respect to the mother- and the world volume:

```
1      /// Create cached matrix to transform to world coordinates
2      const TGeoHMatrix& worldTransformation() const;
3
4      /// Create cached matrix to transform to parent coordinates
5      const TGeoHMatrix& parentTransformation() const;
6
7      /// Transformation from local coordinates of the placed volume to the world system
8      bool localToWorld(const Position& local, Position& global) const;
```

[1] This is a restriction to the possibilities provided by the ROOT implemetation [5] based on experience [3]. If at a later time the need arises the provided alignment interface may be extended to support shape changes.

```
9
10      /// Transformation from local coordinates of the placed volume to the parent system
11      bool localToParent(const Position& local, Position& parent) const;
12
13      /// Transformation from world coordinates of the local placed volume coordinates
14      bool worldToLocal(const Position& global, Position& local) const;
15
16      /// Transformation from world coordinates of the local placed volume coordinates
17      bool parentToLocal(const Position& parent, Position& local) const;
```

it is worth noting that the update of cached information is performed by the *DetElement* objects, other user defined cached information is **not** updated. Such a mechanism shall be provided using update callbacks, which have to be registered to individual *DetElement* entities.
The interface of the *Alignment* structure to access detector alignment parameters is as follows (see also the corresponding header file DD4hep/Alignment.h):

```
1       /// Number of nodes in this branch (=depth of the placement hierarchy from the top level volume)
2       int numNodes() const;
3
4       /// Access the placement of this node
5       PlacedVolume placement()   const;
6       /// Access the placement of the mother of this node
7       PlacedVolume motherPlacement(int level_up = 1)   const;
8
9       /// Access the placement of a node in the chain of placements for this branch
10      PlacedVolume nodePlacement(int level=-1)   const;
11
12      /// Access the currently applied alignment/placement matrix with respect to the world
13      Transform3D toGlobal(int level=-1) const;
14      /// Transform a point from local coordinates of a given level to global coordinates
15      Position toGlobal(const Position& localPoint, int level=-1) const;
16      /// Transform a point from global coordinates to local coordinates of a given level
17      Position globalToLocal(const Position& globalPoint, int level=-1) const;
18
19      /// Access the currently applied alignment/placement matrix with respect to mother volume
20      Transform3D toMother(int level=-1) const;
21
22      /// Access the currently applied alignment/placement matrix (mother to daughter)
23      Transform3D nominal() const;
24
25      /// Access the currently applied correction matrix (delta) (mother to daughter)
26      Transform3D delta() const;
27
28      /// Access the inverse of the currently applied correction matrix (delta) (mother to daughter)
29      Transform3D invDelta() const;
```

- The calls in line 3-7 allow access to the relative position of the *nth.* element in the alignment stack with respect to its next level parent. Element $numNodes() - 1$ denotes the lowest level and element 0 is the world volume. The default argument (-1) addresses the lowest placement in the hierarchy.
- Calls in line 9-10 allow to access/execute transformations from a given level in the placement hierarchy to coordinates in the top level volume (world).
- The call in line 13 allows to transform a global coordinate to the local coordinate system in a given level of the hierarchy. The other two calls of this block support coordinate transformations between local and global coordinate systems.
- The call *toMother* in line 20 returns the local transformation of the node at a given level to the mother's coordinate system.

- The calls in line 16-20 give access to the nominal placement matrix of the realigned node with respect to the parent volume and the changes thereof.

Besides these convenience calls the full interface to the class `TGeoPhysicalNode`, which implements in the ROOT geometry package the support for alignment changes is accessible from the *Alignment* object handle. Further documentation is available from the ROOT documentation .

2.2 Manipulation of Alignment Parameters

There are multiple possibilities to apply alignment parameters:

- The pedestrian way "by hand" using C++ as described in Subsection **??**
- Loading a whole set of misalignment constants from XML, the "poor man's" database. This mechanism is described in Subsection **??**
- Loading a whole set of misalignment constants from a database. This possibility depends heavily on the database and its schema used. A typical use case is to load misalignment constants depending on the experiment conditions at the time the event data were collected. `DDAlign` does not provide an implementation. This possibility here is only mentioned for completeness and will be subject to further developments to support conditions in `DD4hep` .

References

[1] M. Frank et al, "DD4hep: A Detector Description Toolkit for High Energy Physics Experiments", International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013), Amsterdam, Netherlands, 2013, proceedings.

[2] S. Ponce et al., "Detector Description Framework in LHCb", International Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, 2003, proceedings.

[3] C. Parkes, private communications.

[4] M.Frank, "DDG4 - A Simulation Toolkit for High Energy Physics Experiments using Geant4 and the DD4hep Geometry Description".

[5] R.Brun, A.Gheata, M.Gheata, "The ROOT geometry package", Nuclear Instruments and Methods **A** 502 (2003) 676-680.

[6] S. Agostinelli et al., "Geant4 - A Simulation Toolkit", Nuclear Instruments and Methods **A** 506 (2003) 250-303.